

METHOD OF BRIDGING BETWEEN .NET AND JAVA**FIELD**

5 The invention is related to methods of bridging
between different computer languages and platforms,
specifically between Java and Microsoft's .Net framework.

BACKGROUND

10

Web Services are part of the development in
Microsoft's .Net framework for client-server
communications. The .Net specification provides for two
methods of accessing Web Services, SOAP (Simple Object
15 Access Protocol) and .Net Remoting.

20

A Web Service is a unit of application logic providing
data and services to other applications. Applications
access Web Services via ubiquitous Web protocols and data
formats such as HTTP, XML, and SOAP. The .Net platform
from Microsoft represents one system of providing Web
Services.

25

SOAP is an RPC mechanism that uses XML, usually over
HTTP, to allow a client to access a server. SOAP is
beneficial as it is an open-standard XML format allowing
communication between different platforms. There exist

several implementations of SOAP technologies for these .
platforms, as well as Java.

However, SOAP has limitations. The most significant
5 is that the XML format is often not as fast or efficient as
a high-speed binary format.

Also, SOAP lacks support for certain features.
Notably, SOAP does not support activation of lifetime
10 control of remote objects by the client (like DCOM for
Windows). There is also no support for passing objects by
reference and no support for callbacks or events.

SOAP also lacks some of the features provided by .Net.
15 One is the lack of support for additional context
information which is specific to .Net. It is intended that
such information will be used in the future to enable
features such as distributed transactions and additional
security levels.

20

.Net Remoting is an alternative to SOAP. .Net
Remoting is a distributed object protocol that uses binary
or SOAP-based format over TCP or HTTP to allow a client to
access a server. .Net Remoting addresses the limitations

of SOAP by supporting additional features, but at the same time introduces limitations of its own.

The primary limitation of .Net Remoting is that it is
5 specific to the .Net Framework and will only work with
other .Net Frameworks. This presents a particular problem
for developers and organizations that use Java and wish to
combine the .Net Framework with Java.

10 It is an object of this invention to provide a method
of bridging between Java and the .Net Framework. This
method of bridging allows Java clients to use the .Net
Remoting protocol to interact with a Web Service running in
the .Net Framework. This method also allows .Net Framework
15 clients to communicate with Java-based applications using
the .Net Remoting protocol.

SUMMARY

20 The invention is a method for allowing Java objects to
communicate with .Net Remoting objects, with a first step
of receiving metadata information from a .Net Remoting
server on a Java client. Then, Java proxies are generated
from said metadata information, using a Java development
25 tool, with the Java proxies generated by a one-to-one

mapping of .Net classes to Java classes. Finally, the Java proxies are implemented on the Java client, with the method provided solely in Java. Therefore the Java client does not require any .Net components.

5

Preferably, the method also has a Java runtime tool for handling the Java proxies. This Java runtime tool may be capable of independent operation.

10

The invention further includes a method for allowing .Net Remoting objects to communicate with Java objects, with a first step of receiving metadata information from a Java server on a .Net Remoting client. Then, .Net proxies are generated from said metadata information, using a Java development tool, with the .Net proxies generated by a one-to-one mapping of Java classes to .Net classes. Finally, the .Net proxies are implemented on the .Net client, with the method provided solely in CLR metadata. Therefore the .Net client does not require any Java components.

20

The invention also includes a computer program capable of implementing the above methods.

BRIEF DESCRIPTION OF THE DRAWINGS

25

The invention itself both as to organization and

method of operation, as well as additional objects and advantages thereof, will become readily apparent from the following detailed description when read in connection with the accompanying drawings:

5

Figure 1 is a diagram showing the process of generating Java proxies from .NET components;

Figure 2 is a diagram showing the process of
10 generating .NET proxies from Java components;

Figure 3 is a diagram showing the process of Java client to .Net component communication using proxies;

15 Figure 4 is a diagram showing the process of .Net client to Java component communication using proxies.

DETAILED DESCRIPTION

20 The invention is designed to allow Java applications to talk to .Net Remoting objects without any .Net components running on the Java platform and will be referred to as Ja.Net (Java-.Net communication). Ja.Net is a 100% Java-based program designed as a solution to the
25 Java-to-.Net communication problem, unlike other possible

solutions that require the .Net Framework and the Java Virtual Machine to be running on the same machine.

Ja.Net operates in two modes. A development mode
5 generates the necessary proxies to enable communication with a Java or .Net component. A runtime mode allows a user application to use the generated proxies to communicated with a Java or .Net component.

10 .Net Remoting

.Net Remoting is an protocol that facilitates distributed object level computing. Common Language Runtime (CLR), also referred to as .Net Runtime, supports
15 many different languages, including C#, Visual Basic.Net, and C++. Ja.Net allows Java components to appear as CLR components, and CLR components to appear as Java components.

20 .Net Remoting permits the use of a number of different transport protocols and data formats. Currently, HTTP and TCP/IP transport protocols are supported along with SOAP and binary data formatting.

Java to .Net development

The first type of implementation of Ja.Net is one that allows Java objects to talk to .Net Remoting objects. In other words, a Java client is enabled to understand .Net Remoting protocols. Any supported transport protocol and data format supported by .Net Remoting can be used.

The development steps for Java-to-.Net communication are shown in the flowchart of Figure 1. The first step is for the user to specify a name and location for the .Net Remoting server using a GenJava tool to interact with GenService, a continuous running service on the .Net Framework.

The next step 12 is to read the metadata which is related to the server application using GenService. The metadata is then sent (step 14) to the Java client as an XML file.

After receiving the metadata XML file, the Java client must generate (step 16) the necessary Java proxies to parallel the classes and interfaces listed in the metadata. The generated classes have all the methods and properties of the .Net classes. A mapping scheme is used to map the

CLR (.Net) types to Java types during Java proxy generation. An example of CLR to Java type mapping is shown in Table 1.

5

Table 1: CLR-to-Java Mapping

CLR Type	Java Type	Description
void	void	Void
bool/System.Boolean	boolean	True/false value
char/System.Char	char	Unicode 16 bit char
string/System.String	java.lang.String	Unicode String
float32/System.Single	float	IEEE 32-bit float
float64/System.Double	double	IEEE 64-bit float
int8/System.Int8	byte	Signed 8-bit integer
int16/System.Int16	short	Signed 16-bit integer
int32/System.Int32	int	Signed 32-bit integer
int64/System.Int64	long	Signed 64 bit integer
unsigned int8/System.Byte	byte	Unsigned 8-bit integer
unsigned int16/System.UInt16	short	Unsigned 16-bit integer
unsigned int32/System.UInt32	int	Unsigned 32-bit integer
unsigned int64/System.UInt64	long	Unsigned 64-bit integer
System.Object	java.lang.Object	Base class for all objects
System.MarshalByRefObject	java.lang.Object	Base class for all objects passed by reference

There is a direct one-to-one mapping of .Net classes and Java classes. For example, a .Net class called "C" in the namespace "A.B" will generate a Java class named "C" in a Java package named "A.B". More importantly, the class hierarchy is maintained between the .Net and Java classes. This means that Java proxies are generated for the super classes and implemented interfaces of a .Net class as well. However, Java proxies are only generated for those .Net classes with public access. Arrays of class types are also

supported, so that an array of x dimensions in .Net is mapped onto an array of x dimensions in Java.

Marshal by reference classes represent remote objects that return a proxy instead of passing along the object. Each access to the proxy therefore incurs a remote access to the original remote object.

.Net constructors with parameters are also supported.

10 Each public constructor in a .Net class generates two corresponding Java constructors in the Java proxy. For example, the .Net constructor:

15 public Aclass (String s) { }

results in the generation of two Java constructors:

public Aclass (String s) throws RemoteException { }

20 and

public Aclass (String s, String URI, String format, Boolean clientActivated) throws RemoteException { }

25 Both of the Java constructors use the same .Net constructor, however the first Java constructor reads the configuration parameters for the class from the configuration file, whereas the second Java constructor allows the configuration file settings to be overridden.

By specifying any or all of the configuration details listed as parameters in the Java constructor, the settings in the configuration file are overridden by the parameter value.

5

The Java constructors can also throw a RemoteException in case of a communication failure, or in the event of an exception being thrown in the remote constructor itself.

10

For each public method in a .Net class, an equivalent Java method is generated. As with constructors, each method can throw a RemoteException in case of a communication failure, or in the event of an exception being thrown in the remote method itself.

15

Marshal by value classes are used when the class is a container for data. Marshal by value classes are serialized and transmitted. The Java proxy for a marshal by value class contains the fields of the .Net class as public variables, and no methods. Access to the fields does not incur any extra remote access.

20

For a given .Net interface, a Java interface and a Java class are generated. For example, for a .Net interface "Iface", a Java interface "Iface" and a Java

25

class "IfaceProxy" are generated. The Java interface is used by the Java client code, whereas the Java class is used by the runtime to marshal calls through the Java interface. Methods in the interface are mapped as

5 described for the marshal by references classes above.

.Net to Java development

Ja.Net also allows the generation of .Net proxies in

10 order to access a Java Virtual Machine™. The proxy files are C# source files (.cs) that implement classes and class members corresponding to those found in the specified Java classes.

15 The development steps for .Net-to-Java communication are shown in the flowchart of Figure 2. The first step 20 is to specify the names of the Java classes for which a .Net proxy is required using a GenNet tool. This information is provided as metadata (an XML file).

20 The next step 22 is to send the metadata to the .Net client. This can be achieved by using GenService as discussed above for Java-to-.Net communication.

25 When the .Net client receives the metadata XML file, GenService is used to generate (step 24) the necessary C#

classes to parallel the classes and interfaces listed in the metadata. The generated classes have all the methods and properties of the Java classes. A mapping scheme is used to map the Java types to CLR (.Net) types during C# class generation. An example of CLR to Java type mapping is shown in Table 2. Finally, the C# files are compiled (step 26) into a proxy assembly so that the .Net client can access Java while Ja.Net is in runtime mode.

10

Table 2: Java-to-CLR Mapping

Java Type	CLR Type	Description
void	void	Void
boolean	bool	True/false value
char	char	Unicode 16 bit char
java.lang.String	string	Unicode String
float	float32	IEEE 32-bit float
double	float64	IEEE 64-bit float
byte	int8	Signed 8-bit integer
short	int16	Signed 16-bit integer
int	int32	Signed 32-bit integer
long	int64	Signed 64 bit integer
Java.lang.Boolean	bool	True/false value
Java.lang.Char	char	Unicode 16 bit char
Java.lang.Float	float32	IEEE 32-bit float
Java.lang.Double	float64	IEEE 64-bit float
Java.lang.Byte	int8	Signed 8-bit integer
Java.lang.Short	int16	Signed 16-bit integer
Java.lang.Integer	int32	Signed 32-bit integer
Java.lang.Long	int64	Signed 64 bit integer
java.lang.Object	System.MarshalByRefObject	Base class for all objects passed by reference

There is a direct one-to-one mapping of .Net classes and Java classes. Arrays of class types are also

supported, so that an array of x dimensions in Java is mapped onto an array of x dimensions in .Net.

Marshaled classes, constructors and interfaces are all mapped in a similar fashion as described above for Java-to-.Net communication.

Runtime mode

In order to use the proxies generated in the development mode in a user application, a Ja.Net runtime tool is required. The Ja.Net runtime tool provides bi-directional communication between Java and .Net using the proxies generated in the development mode.

Figure 3 shows communication between a Java client and a .Net component 36 using the Ja.Net runtime 34. The Java client 30 accesses the Ja.Net runtime 34 via the Java proxies 32 generated by GenJava. The Ja.Net runtime 34 then converts calls to Java proxies to .Net Remoting calls in order to access the .Net component 36.

Figure 4 shows communication between a .Net client 40 and a Java component 46 using the Ja.Net runtime 44. The .Net client 40 invokes the .Net proxies 42 generated by GenNet. The Ja.Net runtime 44 converts .Net Remoting calls

from the .Net proxies 42 in order to access the Java
component 46.

The Ja.Net development tool is preferably distributed
5 with the Ja.Net runtime tool, to allow for optimization and
verification of applications in development. However, the
Ja.Net runtime tool may be distributed by itself to allow
end users to run applications developed for Ja.Net.

10 Accordingly, while this invention has been described
with reference to illustrative embodiments, this
description is not intended to be construed in a limiting
sense. Various modifications of the illustrative
embodiments, as well as other embodiments of the invention,
15 will be apparent to persons skilled in the art upon
reference to this description. It is therefore
contemplated that the appended claims will cover any such
modifications or embodiments as fall within the scope of
the invention.

20